What Is RCU?

Guest Lecture for University of Cambridge
What is RCU?

Overview

- Mutual Exclusion
- Example Application
- Performance of Synchronization Mechanisms
- Making Software Live With Current (and Future) Hardware
- Implementing RCU
- RCU Grace Periods: Conceptual and Graphical Views
- Performance
- RCU Area of Applicability
- Summary
What is RCU?

Mutual Exclusion
What is RCU?

Mutual Exclusion

- What mechanisms can enforce mutual exclusion?
What is RCU?

Example Application
What is RCU?

Example Application

- Schrödinger wants to construct an in-memory database for the animals in his zoo (example in upcoming ACM Queue)
  - Births result in insertions, deaths in deletions
  - Queries from those interested in Schrödinger's animals
  - Lots of short-lived animals such as mice: High update rate
  - Great interest in Schrödinger's cat (perhaps queries from mice?)
What is RCU?

Example Application

- Schrödinger wants to construct an in-memory database for the animals in his zoo (example in upcoming ACM Queue)
  - Births result in insertions, deaths in deletions
  - Queries from those interested in Schrödinger's animals
  - Lots of short-lived animals such as mice: High update rate
  - Great interest in Schrödinger's cat (perhaps queries from mice?)

- Simple approach: chained hash table with per-bucket locking
What is RCU?

Example Application

- Schrödinger wants to construct an in-memory database for the animals in his zoo (example in upcoming ACM Queue)
  - Births result in insertions, deaths in deletions
  - Queries from those interested in Schrödinger's animals
  - Lots of short-lived animals such as mice: High update rate
  - Great interest in Schrödinger's cat (perhaps queries from mice?)

- Simple approach: chained hash table with per-bucket locking

![Diagram of chained hash table with per-bucket locking]

Will holding this lock prevent the cat from dying?
What is RCU?

Read-Only Bucket-Locked Hash Table Performance

Why the dropoff??

2GHz Intel Xeon Westmere-EX, 1024 hash buckets
What is RCU?

Varying Number of Hash Buckets

2GHz Intel Xeon Westmere-EX

Still a dropoff...

16384
8192
4096
2048

Lookups per Millisecond

Number of CPUs/Threads
What is RCU?

NUMA Effects???

- `/sys/devices/system/cpu/cpu0/cache/index0/shared_cpu_list: -0,32`
- `/sys/devices/system/cpu/cpu0/cache/index1/shared_cpu_list: -0,32`
- `/sys/devices/system/cpu/cpu0/cache/index2/shared_cpu_list: -0-7,32-39`

- Two hardware threads per core, eight cores per socket
- Try using only one CPU per socket: CPUs 0, 8, 16, and 24
What is RCU?

Bucket-Locked Hash Performance: 1 CPU/Socket

This is not the sort of scalability Schrödinger requires!!!
What is RCU?

Performance of Synchronization Mechanisms
What is RCU?

Performance of Synchronization Mechanisms

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost (ns)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock period</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>“Best-case” CAS</td>
<td>12.2</td>
<td>33.8</td>
</tr>
<tr>
<td>Best-case lock</td>
<td>25.6</td>
<td>71.2</td>
</tr>
<tr>
<td>Single cache miss</td>
<td>12.9</td>
<td>35.8</td>
</tr>
<tr>
<td>CAS cache miss</td>
<td>7.0</td>
<td>19.4</td>
</tr>
<tr>
<td>Single cache miss (off-core)</td>
<td>31.2</td>
<td>86.6</td>
</tr>
<tr>
<td>CAS cache miss (off-core)</td>
<td>31.2</td>
<td>86.5</td>
</tr>
<tr>
<td>Single cache miss (off-socket)</td>
<td>92.4</td>
<td>256.7</td>
</tr>
<tr>
<td>CAS cache miss (off-socket)</td>
<td>95.9</td>
<td>266.4</td>
</tr>
</tbody>
</table>

And these are best-case values!!! (Why?)
What is RCU?

Why All These Low-Level Details???
Why All These Low-Level Details??

- Would you trust a bridge designed by someone who did not understand strengths of materials?
- Or a ship designed by someone who did not understand the steel-alloy transition temperatures?
- Or a house designed by someone who did not understand that unfinished wood rots when wet?
- Or a car designed by someone who did not understand the corrosion properties of the metals used in the exhaust system?
- Or a space shuttle designed by someone who did not understand the temperature limitations of O-rings?
What is RCU?

**Why All These Low-Level Details??**

- Would you trust a bridge designed by someone who did not understand strengths of materials?
- Or a ship designed by someone who did not understand the steel-alloy transition temperatures?
- Or a house designed by someone who did not understand that unfinished wood rots when wet?
- Or a car designed by someone who did not understand the corrosion properties of the metals used in the exhaust system?
- Or a space shuttle designed by someone who did not understand the temperature limitations of O-rings?

- So why trust algorithms from someone ignorant of the properties of the underlying hardware???
What is RCU?

But What Do The Operation Timings Really Mean???
What is RCU?

But What Do The Operation Timings Really Mean???

- Single-instruction critical sections protected by multiple locks

Uncontended

Contended, No Spinning

So, what does this mean?
What is RCU?

But What Do The Operation Timings Really Mean???

- Single-instruction critical sections protected by multiple locks

Uncontended:
- 256.7 cycles

Contended, No Spinning:
- 256.7 cycles

258 CPUs to break even with single CPU!

514 CPUs to break even with single CPU!!!
What is RCU?

But What Do The Operation Timings Really Mean???

- Single-instruction critical sections protected by multiple locks

Uncontended

- 256.7 cycles
- 1 cycle

Contended, No Spinning

- 256.7 cycles
- 1 cycle
- 256.7 cycles

Contended, Spinning

- ??? cycles
- 256.7 cycles
- 256.7 cycles

258 CPUs to break even with single CPU!
514 CPUs to break even with single CPU!!!
What is RCU?

Reader-Writer Locks Are Even Worse!
What is RCU?

Reader-Writer Locks Are Even Worse!

CPU 0

Acquire

266.4 cycles

Critical Section

1 cycle

Wait for Lock Data

266.4 cycles

Release

266.4 cycles

200.0 cycles

CPU 1

Spin

Acquire

Critical Section

800 CPUs to break even with a single CPU!!!
But What About Scaling With Atomic Operations?
What is RCU?

If You Think Single Atomic is Expensive, Try Lots!!!

2GHz Intel Xeon Westmere-EX
What is RCU?

Why So Slow???
What is RCU?

System Hardware Structure and Laws of Physics

Electrons move at 0.03C to 0.3C in transistors and, so lots of waiting. 3D???
What is RCU?

Atomic Increment of Global Variable

Lots and Lots of Latency!!!
What is RCU?

Atomic Increment of Per-CPU Counter

Little Latency, Lots of Increments at Core Clock Rate
What is RCU?

Can't The Hardware Do Better Than This???
What is RCU?

**HW-Assist Atomic Increment of Global Variable**

SGI systems used this approach in the 1990s, expect modern CPUs to optimize. Still not as good as per-CPU counters.
What is RCU?

**HW-Assist Atomic Increment of Global Variable**

Put an ALU near memory to avoid slowdowns due to latency. Still not as good as per-CPU counters.
What is RCU?

Problem With Physics #1: Finite Speed of Light
What is RCU?

Problem With Physics #2: Atomic Nature of Matter

I feel so fat!

No complaints for eons, and now, suddenly, we're too $#@* big?!

And our dielectric constant isn't **big enough** for them! They can go find some other $#@* atom! Sheesh!
What is RCU?

How Can Software Live With This Hardware???
What is RCU?

Design Principle: Avoid Bottlenecks

Only one of something: bad for performance and scalability. Also typically results in high complexity.
What is RCU?

**Design Principle: Avoid Bottlenecks**

Many instances of something good!

Avoiding tightly coupled interactions is an excellent way to avoid bugs.
What is RCU?

**Design Principle: Avoid Bottlenecks**

Many instances of something good!
Avoiding tightly coupled interactions is an excellent way to avoid bugs.
But NUMA effects defeated this for per-bucket locking!!!
### Design Principle: Avoid Expensive Operations

16-CPU 2.8GHz Intel X5550 (Nehalem) System

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost (ns)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock period</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>“Best-case” CAS</td>
<td>12.2</td>
<td>33.8</td>
</tr>
<tr>
<td>Best-case lock</td>
<td>25.6</td>
<td>71.2</td>
</tr>
<tr>
<td>Single cache miss</td>
<td>12.9</td>
<td>35.8</td>
</tr>
<tr>
<td>CAS cache miss</td>
<td>7.0</td>
<td>19.4</td>
</tr>
<tr>
<td>Single cache miss (off-core)</td>
<td>31.2</td>
<td>86.6</td>
</tr>
<tr>
<td>CAS cache miss (off-core)</td>
<td>31.2</td>
<td>86.5</td>
</tr>
<tr>
<td>Single cache miss (off-socket)</td>
<td>92.4</td>
<td>256.7</td>
</tr>
<tr>
<td>CAS cache miss (off-socket)</td>
<td>95.9</td>
<td>266.4</td>
</tr>
</tbody>
</table>

**Need to be here! (Partitioning/RCU)**

Heavily optimized reader-writer lock might get here for readers (but too bad about those poor writers...)

**Typical synchronization mechanisms do this a lot**
What is RCU?

Design Principle: Get Your Money's Worth

- If synchronization is expensive, use large critical sections
  - On Nehalem, off-socket CAS costs about 260 cycles
    - So instead of a single-cycle critical section, have a 26000-cycle critical section, reducing synchronization overhead to about 1%
- Of course, we also need to keep contention low, which usually means we want short critical sections
  - Resolve this by applying parallelism at as high a level as possible
  - Parallelize entire applications rather than low-level algorithms!
- This does not work for Schrödinger: The overhead of hash-table operations is too low
  - Which is precisely why we selected hash tables in the first place!!!
What is RCU?

Design Principle: Avoid Mutual Exclusion!!!

CPU 0: Reader | Reader | Reader

CPU 1: Reader | Reader | Reader | Reader

CPU 2: Reader | Reader

CPU 3: Reader | Reader | Spin | Updater | Reader

Dead Time!!!
### What is RCU?

**Design Principle: Avoiding Mutual Exclusion**

<table>
<thead>
<tr>
<th>CPU 0</th>
<th>Reader</th>
<th>Reader</th>
<th>Reader</th>
<th>Reader</th>
<th>Reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 1</td>
<td>Reader</td>
<td>Reader</td>
<td>Reader</td>
<td>Reader</td>
<td>Reader</td>
</tr>
<tr>
<td>CPU 2</td>
<td>Reader</td>
<td>Reader</td>
<td>Reader</td>
<td>Reader</td>
<td>Reader</td>
</tr>
<tr>
<td>CPU 3</td>
<td>Reader</td>
<td>Reader</td>
<td>Updater</td>
<td>Reader</td>
<td>Reader</td>
</tr>
</tbody>
</table>

*No Dead Time!*
What is RCU?

But How Can This Possibly Be Implemented???
What is RCU?

Implementing Read-Copy Update (RCU)

- Lightest-weight conceivable read-side primitives
What is RCU?

Implementing Read-Copy Update (RCU)

- Lightest-weight conceivable read-side primitives
  - /* Assume non-preemptible (run-to-block) environment. */
  - #define rcu_read_lock()
  - #define rcu_read_unlock()
What is RCU?

Implementing Read-Copy Update (RCU)

 Lightest-weight conceivable read-side primitives
  – /* Assume non-preemptible (run-to-block) environment. */
  – #define rcu_read_lock()
  – #define rcu_read_unlock()

 Best possible performance, scalability, real-time response, wait-freedom, and energy efficiency
What is RCU?

Implementing Read-Copy Update (RCU)

- Lightest-weight conceivable read-side primitives
  - /* Assume non-preemptible (run-to-block) environment. */
  - #define rcu_read_lock()
  - #define rcu_read_unlock()

- Best possible performance, scalability, real-time response, wait-freedom, and energy efficiency

- How can something that does not affect machine state possibly be used as a synchronization primitive???
What is RCU?

What Is RCU?

- Publishing of new data
- Subscribing to the current version of data
- Waiting for pre-existing RCU readers: Avoid disrupting readers by maintaining multiple versions of the data
  - Each reader continues traversing its copy of the data while a new copy might be being created concurrently by each updater *
  - Hence the name read-copy update, or RCU
  - Once all pre-existing RCU readers are done with them, old versions of the data may be discarded

* This expansion provided by Jonathan Walpole
What is RCU?

Publication of And Subscription to New Data

Key:
- **Dangerous for updates**: all readers can access
- **Still dangerous for updates**: pre-existing readers can access (next slide)
- **Safe for updates**: inaccessible to all readers

```c
kmalloc()
cptr
-tmp
->a=?
->b=?
->c=?

initialization

cptr
-tmp
->a=1
->b=2
->c=3

= cptr

reader

cptr = tmp
->a=1
->b=2
->c=3
```
What is RCU?

Memory Ordering: Mischief From Compiler and CPU
What is RCU?

Memory Ordering: Mischief From Compiler and CPU

- Original updater code:
  ```c
  p = malloc(sizeof(*p));
  p->a = 1;
  p->b = 2;
  p->c = 3;
  cptr = p;
  ```

- Original reader code:
  ```c
  p = cptr;
  foo(p->a, p->b, p->c);
  ```

- Mischievous updater code:
  ```c
  p = malloc(sizeof(*p));
  cptr = p;
  p->a = 1;
  p->b = 2;
  p->c = 3;
  ```

- Mischievous reader code:
  ```c
  retry:
  p = guess(cptr);
  foo(p->a, p->b, p->c);
  if (p != cptr)
    goto retry;
  ```
What is RCU?

Memory Ordering: Mischief From Compiler and CPU

- Original updater code:
  ```c
  p = malloc(sizeof(*p));
  p->a = 1;
  p->b = 2;
  p->c = 3;
  cptr = p;
  ```

- Original reader code:
  ```c
  p = cptr;
  foo(p->a, p->b, p->c);
  ```

- Mischievous updater code:
  ```c
  p = malloc(sizeof(*p));
  cptr = p;
  p->a = 1;
  p->b = 2;
  p->c = 3;
  ```

- Mischievous reader code:
  ```c
  retry:
  p = guess(cptr);
  foo(p->a, p->b, p->c);
  if (p != cptr)
    goto retry;
  ```

But don't take my word for it on HW value speculation:
http://www.openvms.compaq.com/wizard/wiz_2637.html
What is RCU?

Preventing Memory-Order Mischief

- Updater uses `rcu_assign_pointer()` to publish pointer:
  ```c
  #define rcu_assign_pointer(p, v) \
  ({ \
    smp_wmb(); /* SMP Write Memory Barrier */ \ 
    (p) = (v); \
  })
  ```

- Reader uses `rcu_dereference()` to subscribe to pointer:
  ```c
  #define rcu_dereference(p) \
  ({ \
    typeof(p) _p1 = (*(volatile typeof(p)*)&(p)); \ 
    smp_read_barrier_depends(); \ 
    _p1; \
  })
  ```

- The Linux-kernel definitions are more ornate: Debugging code
What is RCU?

Preventing Memory-Order Mischief

- “Memory-order-mischief proof” updater code:
  
  ```c
  p = malloc(sizeof(*p));
p->a = 1;
p->b = 2;
p->c = 3;
rcu_assign_pointer(cptr, p);
  ```

- “Memory-order-mischief proof” reader code:

  ```c
  p = rcu_dereference(cptr);
  foo(p->a, p->b, p->c);
  ```
What is RCU?

Publication of And Subscription to New Data

Key:
- Dangerous for updates: all readers can access
- Still dangerous for updates: pre-existing readers can access (next slide)
- Safe for updates: inaccessible to all readers

- cptr
  - malloc()
  - tmp

- cptr
  - initialization
  - ->a=?
  - ->b=?
  - ->c=?

- cptr
  - rcu_assign_pointer(cptr,p)
  - ->a=1
  - ->b=2
  - ->c=3

- tmp

But if all we do is add, we have a big memory leak!!!
What is RCU?

RCU Removal From Linked List

- Combines waiting for readers and multiple versions:
  - Writer removes the cat's element from the list (list_del_rcu())
  - Writer waits for all readers to finish (synchronize_rcu())
  - Writer can then free the cat's element (kfree())
What is RCU?

RCU Removal From Linked List

- Combines waiting for readers and multiple versions:
  - Writer removes the cat's element from the list (list_del_rcu())
  - Writer waits for all readers to finish (synchronize_rcu())
  - Writer can then free the cat's element (kfree())

But how can software deal with two different versions simultaneously???
What is RCU?

Two Different Versions Simultaneously??

I think the poor thing has expired.

No!

Where there is a brain-wave, there is a way!
What is RCU?

RCU Removal From Linked List

- Combines waiting for readers and multiple versions:
  - Writer removes the cat's element from the list (list_del_rcu())
  - Writer waits for all readers to finish (synchronize_rcu())
  - Writer can then free the cat's element (kfree())

But if readers leave no trace in memory, how can we possibly tell when they are done???
What is RCU?

How Can RCU Tell When Readers Are Done???
What is RCU?

How Can RCU Tell When Readers Are Done???

That is, without re-introducing all of the overhead and latency inherent to other synchronization mechanisms...
But First, Some RCU Nomenclature

- **RCU read-side critical section**
  - Begins with `rcu_read_lock()`, ends with `rcu_read_unlock()`, and may contain `rcu_dereference()`

- **Quiescent state**
  - Any code that is not in an RCU read-side critical section

- **Extended quiescent state**
  - Quiescent state that persists for a significant time period

- **RCU grace period**
  - Time period when every thread was in at least one quiescent state
What is RCU?

But First, Some RCU Nomenclature

- **RCU read-side critical section**
  - Begins with `rcu_read_lock()`, ends with `rcu_read_unlock()`, and may contain `rcu_dereference()`

- **Quiescent state**
  - Any code that is not in an RCU read-side critical section

- **Extended quiescent state**
  - Quiescent state that persists for a significant time period

- **RCU grace period**
  - Time period when every thread was in at least one quiescent state

- OK, names are nice, but how can you possibly implement this???
What is RCU?

Waiting for Pre-Existing Readers: QSBR

- Non-preemptive environment (CONFIG_PREEMPT=n)
  - RCU readers are not permitted to block
  - Same rule as for tasks holding spinlocks
What is RCU?

Waiting for Pre-Existing Readers: QSBR

- Non-preemptive environment (CONFIG_PREEMPT=n)
  - RCU readers are not permitted to block
  - Same rule as for tasks holding spinlocks

- CPU context switch means all that CPU's readers are done

**Grace period** ends after all CPUs execute a context switch
What is RCU?

Synchronization Without Changing Machine State???

- But rcu_read_lock() does not need to change machine state
  - Instead, it acts on the developer, who must avoid blocking within RCU read-side critical sections
  - Or, more generally, avoid quiescent states within RCU read-side critical sections
What is RCU?

Synchronization Without Changing Machine State???

- But rcu_read_lock() does not need to change machine state
  - Instead, it acts on the developer, who must avoid blocking within RCU read-side critical sections
  - Or, more generally, avoid quiescent states within RCU read-side critical sections

- RCU is therefore synchronization via social engineering
What is RCU?

Synchronization Without Changing Machine State???

- But rcu_read_lock() does not need to change machine state
  - Instead, it acts on the developer, who must avoid blocking within RCU read-side critical sections
  - Or, more generally, avoid quiescent states within RCU read-side critical sections

- RCU is therefore synchronization via social engineering

- As are all other synchronization mechanisms:
  - “Avoid data races”
  - “Protect specified variables with the corresponding lock”
  - “Access shared variables only within transactions”
What is RCU?

Toy Implementation of RCU: 20 Lines of Code

- **Read-side primitives:**
  - #define rcu_read_lock()
  - #define rcu_read_unlock()
  - #define rcu_dereference(p) \\
    ({{ typeof(p) _p1 = (*((volatile typeof(p)*)&(p))); \\
        smp_read_barrier_depends(); \\
        _p1; \\
    })

- **Update-side primitives**
  - #define rcu_assign_pointer(p, v) \\
    ({{ smp_wmb(); \\
        (p) = (v); \\
    })

void synchronize_rcu(void)
{
    int cpu;

    for_each_online_cpu(cpu)
        run_on(cpu);
}
What is RCU?

**Toy Implementation of RCU: 20 Lines of Code, Full Read-Side Performance!!!**

- **Read-side primitives**
  ```c
  #define rcu_read_lock()
  #define rcu_read_unlock()
  #define rcu_dereference(p) 
  ({
      typeof(p) _p1 = (*(volatile typeof(p)*)&(p)); \
      smp_read_barrier_depends(); \
      _p1;
  })
  ```

- **Update-side primitives**
  ```c
  #define rcu_assign_pointer(p, v) 
  ({
      smp_wmb(); \
      (p) = (v);
  })
  ```

  ```c
  void synchronize_rcu(void)
  {
      int cpu;

      for_each_online_cpu(cpu)
          run_on(cpu);
  }
  ```

Only 9 of which are needed on sequentially consistent systems...
And some people still insist that RCU is complicated... ;-)
What is RCU?

RCU Usage: Readers

- Pointer to RCU-protected object guaranteed to exist throughout RCU read-side critical section
  
  ```c
  rcu_read_lock(); /* Start critical section. */
  p = rcu_dereference(cptr);
  /* *p guaranteed to exist. */
  do_something_with(p);
  rcu_read_unlock(); /* End critical section. */
  /* *p might be freed!!! */
  ```

- The `rcu_read_lock()`, `rcu_dereference()` and `rcu_read_unlock()` primitives are very light weight

- However, updaters must take care...
What is RCU?

RCU Usage: Updaters

- Updaters must wait for an *RCU grace period* to elapse between making something inaccessible to readers and freeing it

  ```c
  spin_lock(&updater_lock);
  q = cptr;
  rcu_assign_pointer(cptr, new_p);
  spin_unlock(&updater_lock);
  synchronize_rcu(); /* Wait for grace period. */
  kfree(q);
  ```

- RCU grace period waits for all pre-exiting readers to complete their RCU read-side critical sections
What is RCU?

Complex Atomic-To-Reader Updates
What is RCU?

RCU Replacement Of Item In Linked List

1 Version
boa
   \arrow{1 Version} \rightarrow cat
   \arrow{1 Version} \rightarrow gnu

1 Version
boa
   \arrow{1 Version} \rightarrow ?
   \arrow{1 Version} \rightarrow copy

1 Version
boa
   \arrow{1 Version} \rightarrow cat
   \arrow{1 Version} \rightarrow gnu

1 Version
boa
   \arrow{1 Version} \rightarrow cat'
   \arrow{1 Version} \rightarrow gnu

1 Version
boa
   \arrow{1 Version} \rightarrow list_replace_rcu()

1 Version
boa
   \arrow{1 Version} \rightarrow cat'
   \arrow{1 Version} \rightarrow synchronize_rcu()

1 Version
boa
   \arrow{1 Version} \rightarrow cat
   \arrow{1 Version} \rightarrow gnu

1 Version
boa
   \arrow{1 Version} \rightarrow kfree()
What is RCU?

RCU Grace Periods: Conceptual and Graphical Views
What is RCU?

RCU Grace Periods: A Conceptual View

- **RCU read-side critical section**
  - Begins with `rcu_read_lock()`, ends with `rcu_read_unlock()`, and may contain `rcu_dereference()`

- **Quiescent state**
  - Any code that is not in an RCU read-side critical section

- **Extended quiescent state**
  - Quiescent state that persists for a significant time period

- **RCU grace period**
  - Time period when every thread is in at least one quiescent state
  - Ends when all pre-existing readers complete
  - Guaranteed to complete in finite time iff all RCU read-side critical sections are of finite duration

- But what happens if you try to extend an RCU read-side critical section across a grace period?
What is RCU?

RCU Grace Periods: A Graphical View

So what happens if you try to extend an RCU read-side critical section across a grace period?
What is RCU?

RCU Grace Period: A Self-Repairing Graphical View

A grace period is not permitted to end until all pre-existing readers have completed.
What is RCU?

RCU Grace Period: A Lazy Graphical View

But it is OK for RCU to be lazy and allow a grace period to extend longer than necessary.
What is RCU?

RCU Grace Period: A *Really* Lazy Graphical View

And it is also OK for RCU to be even more lazy and start a grace period later than necessary. But why is this useful?

`synchronize_rcu()`
What is RCU?

RCU Grace Period: A Usefully Lazy Graphical View

Starting a grace period late can allow it to serve multiple updates, decreasing the per-update RCU overhead. But...

- Reader
- Reader
- Reader
- Reader
- Reader

synchronize_rcu()
What is RCU?

The Costs and Benefits of Laziness

- Starting the grace period later increases the number of updates per grace period, reducing the per-update overhead
- Delaying the end of the grace period increases grace-period latency
- Increasing the number of updates per grace period increases the memory usage
  - Therefore, starting grace periods late is a good tradeoff if memory is cheap and communication is expense, as is the case in modern multicore systems
  - And if real-time threads avoid waiting for grace periods to complete
  - However...
What is RCU?

RCU Grace Period: A Too-Lazy Graphical View

And it is OK for the system to complain (or even abort) if a grace period extends too long. Too-long grace periods are likely to result in death by memory exhaustion anyway.
What is RCU?

RCU Asynchronous Grace-Period Detection
What is RCU?

RCU Asynchronous Grace-Period Detection

- The `call_rcu()` function registers an RCU callback, which is invoked after a subsequent grace period elapses

- **API:**

  ```c
  call_rcu(struct rcu_head head,
           void (*func)(struct rcu_head *rcu));
  ```

- **The `rcu_head` structure:**

  ```c
  struct rcu_head {
    struct rcu_head *next;
    void (*func)(struct rcu_head *rcu);
  };
  ```

- The `rcu_head` structure is normally embedded within the RCU-protected data structure
What is RCU?

RCU Grace Period: An Asynchronous Graphical View

```
call_rcu(&p->rcu, func);
call_rcu(&p->rcu, func);
func(&p->rcu);
```
What is RCU?

Performance
What is RCU?

Theoretical Performance

RCU (wait-free)

Uncontended

71.2 cycles

1 cycle

Full performance, linear scaling, real-time response

73 CPUs to break even with a single CPU!

Contended, No Spinning

71.2 cycles

1 cycle

71.2 cycles

144 CPUs to break even with a single CPU!!!
What is RCU?

Measured Performance
What is RCU?

Schrödinger's Zoo: Read-Only

RCU and hazard pointers scale quite well!!!
What is RCU?

Schrödinger's Zoo: Read-Only Cat-Heavy Workload

RCU handles locality, hazard pointers not bad, bucket locking horrible!
### Schrödinger's Zoo: Reads and Updates

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Reads</th>
<th>Failed Reads</th>
<th>Cat Reads</th>
<th>Adds</th>
<th>Deletes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Locking</td>
<td>799</td>
<td>80</td>
<td>639</td>
<td>77</td>
<td>77</td>
</tr>
<tr>
<td>Per-Bucket Locking</td>
<td>13,555</td>
<td>6,177</td>
<td>1,197</td>
<td>5,370</td>
<td>5,370</td>
</tr>
<tr>
<td>Hazard Pointers</td>
<td>41,011</td>
<td>6,982</td>
<td>27,059</td>
<td>4,860</td>
<td>4,860</td>
</tr>
<tr>
<td>RCU</td>
<td>85,906</td>
<td>13,022</td>
<td>59,873</td>
<td>2,440</td>
<td>2,440</td>
</tr>
</tbody>
</table>
What is RCU?

Real-Time Response to Changes
What is RCU?

RCU vs. Reader-Writer-Lock Real-Time Latency

External Event

RCU Latency

rwlock latency

RCU reader

RCU reader

RCU reader

RCU updater

RCU reader

RCU reader

RCU reader

spin

RWlock Reader

RWlock Reader

spin

RWlock Writer

RWlock Reader

spin

RWlock Reader

RCU Reader

RCU Reader

RCU Reader

External Event

RCU Latency

rwlock Latency
What is RCU?

RCU Performance: “Free is a Very Good Price!!!”
What is RCU?

RCU Performance: “Free is a Very Good Price!!!”
And Nothing Is Faster Than Doing Nothing!!!
What is RCU?

RCU Area of Applicability

Read-Mostly, Stale & Inconsistent Data OK (RCU Works Great!!)

Read-Mostly, Need Consistent Data (RCU Works OK)

Read-Write, Need Consistent Data (RCU Might Be OK...)

Update-Mostly, Need Consistent Data (RCU is Really Unlikely to be the Right Tool For The Job, But It Can:
(1) Provide Existence Guarantees For Update-Friendly Mechanisms
(2) Provide Wait-Free Read-Side Primitives for Real-Time Use)

Schrodinger's zoo is in blue: Can't tell exactly when an animal is born or dies anyway! Plus, no lock you can hold will prevent an animal's death...
What is RCU?

RCU Applicability to the Linux Kernel
What is RCU?

Summary
What is RCU?

Summary

- Synchronization overhead is a big issue for parallel programs
- Straightforward design techniques can avoid this overhead
  - Partition the problem: “Many instances of something good!”
  - Avoid expensive operations
  - Avoid mutual exclusion
- RCU is part of the solution
  - Excellent for read-mostly data where staleness and inconsistency OK
  - Good for read-mostly data where consistency is required
  - Can be OK for read-write data where consistency is required
  - Might not be best for update-mostly consistency-required data
  - Used heavily in the Linux kernel
- Much more information on RCU is available...
What is RCU?

Graphical Summary

Not only are they lazy, they get more work done than I do!

“yawn”

TM

Locking

Hazard Pointers

ZZZ

RCU

ZZZ

ZZZ
What is RCU?

To Probe Further:

- https://queue.acm.org/detail.cfm?id=2488549
  - “Structured Deferral: Synchronization via Procrastination” (also in July 2013 CACM)
  - “User-Level Implementations of Read-Copy Update”
- git://lttng.org/userspace-rcu.git (User-space RCU git tree)
  - Applying RCU and weighted-balance tree to Linux mmap_sem.
  - RCU-protected resizable hash tables, both in kernel and user space
  - Combining RCU and software transactional memory
- http://wiki.cs.pdx.edu/rp/: Relativistic programming, a generalization of RCU
- http://lwn.net/Articles/262464/, http://lwn.net/Articles/263130/, http://lwn.net/Articles/264090/
  - “What is RCU?” Series
  - RCU motivation, implementations, usage patterns, performance (micro+sys)
  - System-level performance for SELinux workload: >500x improvement
  - Comparison of RCU and NBS (later appeared in JPDC)
- http://doi.acm.org/10.1145/1400097.1400099
  - History of RCU in Linux (Linux changed RCU more than vice versa)
  - Harvard University class notes on RCU (Courtesy of Eddie Koher)
What is RCU?

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
- Credits:
  - This material is based upon work supported by the National Science Foundation under Grant No. CNS-0719851.
  - Joint work with Mathieu Desnoyers, Alan Stern, Michel Dagenais, Manish Gupta, Maged Michael, Phil Howard, Joshua Triplett, Jonathan Walpole, and the Linux kernel community.
  - Additional reviewers: Carsten Weinhold and Mingming Cao.
Questions?

Use the right tool for the job!!!

Image copyright © 2004 Melissa McKenney